FIFO

May 20, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

Standard FIFO with multiple options. The FIFO uses a similar interface to the Xilinx FIFO. It also emulates the Xilinx FIFO bugs and all. This is NOT dependent on Xilinx FPGA's and can be used on any FPGA supporting the Verilog block ram style primitive.

1.2 Dependencies

The following are the dependencies of the cores.

- fusesoc 2.X
- iverilog (simulation)
- cocotb (simulation)

1.2.1 fusesoc_info Depenecies

- dep
  - AFRL:utility:helper:1.0.0
  - AFRL:ram:dc_block_ram:1.0.0
- dep_tb
  - AFRL:simulation:fifo_stimulator
  - AFRL:simulation:clock_stimulator
  - AFRL:utility:sim_helper

1.3 In a Project

Simply use this core between a sink and source devices. This buffer data from one bus to another. Check the code to see if others will work correctly.

2 Architecture

This FIFO is made for three modules. They are the FIFO pipe, FIFO control, and dual clock RAM. The combination of these three provide the FIFO module. Having it made this way allows for future modules to be customized and brought in to change the FIFO's behavior. The

current modules emulate the Xilinx FIFO IP core availble in Vivado 2018 and up.

FIFO pipe creates a set of pipeline registers for the data interfaces. This helps fix timing issues in the core and pipeline depth can be changed via parameters.

FIFO control is the heart of the core when it comes to how it responds. The logic in the core is designed to emulate the Xilinx FIFO IP.

Dual clock RAM is a universal block RAM core.

Please see 5 for more information.

3 Building

The FIFO core is written in Verilog 2001. They should synthesize in any modern FPGA software. The core comes as a fusesoc packaged core and can be included in any other core. Be sure to make sure you have meet the dependencies listed in the previous section. Linting is performed by the lint target using verible.

3.1 fusesoc

Fusesoc is a system for building FPGA software without relying on the internal project management of the tool. Avoiding vendor lock in to Vivado or Quartus. These cores, when included in a project, can be easily integrated and targets created based upon the end developer needs. The core by itself is not a part of a system and should be integrated into a fusesoc based system. Simulations are setup to use fusesoc and are a part of its targets.

3.2 Source Files

3.2.1 fusesoc_info File List

- src
  - src/fifo.v
  - src/fifo_ctrl.v
  - src/fifo_pipe.v
- tb
  - 'tb/tb_fifo.v': 'file_type': 'verilogSource'
- tb_cocotb

- 'tb/tb_cocotb.py': 'file_type': 'user', 'copyto': '.'
- 'tb/tb_cocotb.v': 'file_type': 'verilogSource'

- constr
  - 'tool_vivado ? (constr/fifo_constr.tcl)': 'file_type': 'SDC'

3.3 Targets

3.3.1 fusesoc_info Targets

- default

  Info: Default for IP intergration.

- lint

  Info: Lint with Verible

- sim

  Info: Constant data value with file check.

- sim_rand_data

  Info: Feed random data input with file check

- sim_rand_ready_rand_data

  Info: Feed random data input, and randomize the read ready on the output. Perform output file check.

- sim_8bit_count_data

  Info: Feed a counter data as input, perform file check.

- sim_cocotb

  Info: Cocotb unit tests

3.4 Directory Guide

Below highlights important folders from the root of the directory.

1. **docs** Contains all documentation related to this project.
   - **manual** Contains user manual and github page that are generated from the latex sources.
2. **src** Contains source files for the core
3. **tb** Contains test bench files for iverilog and cocotb
   - **cocotb** testbench files

4 Simulation

There are a few different simulations that can be run for this core.

4.1 iverilog

All simulation targets that do NOT have cocotb in the name use a verilog test bench with verilog stimulus components. These all read in a file and then write a file that has been processed by the FIFO. Then the input and output file are compared with a MD5 sum to check that they match. If they do not match then the test has failed. All of these tests provide fst output files for viewing the waveform in the there target build folder.

4.2 cocotb

To use the cocotb tests you must install the following python libraries.

```
$ pip install cocotb
$ pip install cocotbext-fifo
```

Then you must use the cocotb sim target. In this case it is sim_cocotb. This target can be run with various bus and fifo parameters.

```
$ fusesoc run --target sim_cocotb AFRL:buffer:fifo
    ↪ :1.2.0 --BUS_WIDTH=8 --FIFO_DEPTH=32
```

The following is an example command to run through various parameters without typing them one by one.

```
$ for i in {1..32}; do sleep 5; export RY=$(($RANDOM
    ↪ %32+1)); fusesoc run --target sim_cocotb AFRL:
    ↪ buffer:axis_fifo:1.0.0 --BUS_WIDTH=$i --
    ↪ FIFO_DEPTH=$RY; echo "BUS␣WIDTH:" $i "FIFO␣DEPTH:
    ↪ " $RY; done
```

5 Module Documentation

There is a single async module for this core.

- **FIFO** FIFO will buffer data from input to output.
- **FIFO_PIPE** FIFO_PIPE will provide a pipeline for timing issues.
- **FIFO_CONTROL** FIFO_CONTROL emulates the Xilinx FIFO IP interface and its behavior.
- **FIFO_COCOTB PYTHON** Cocotb python test bench.
- **FIFO_COCOTB VERILOG** Cocotb verilog wrapper.

The next sections document the modules.

fifo.v

AUTHORS

JAY CONVERTINO

DATES

2021/06/29

INFORMATION

Brief

Wrapper to tie together fifo_ctrl, fifo_mem, and fifo_pipe.

License MIT

fifo

```
module fifo #(
parameter
FIFO_DEPTH
 =
256,
parameter
BYTE_WIDTH
 =
1,
parameter
COUNT_WIDTH
 =
8,
parameter
```

```
FWFT
 =
0,
parameter
RD_SYNC_DEPTH
 =
0,
parameter
WR_SYNC_DEPTH
 =
0,
parameter
DC_SYNC_DEPTH
 =
0,
parameter
COUNT_DELAY
 =
1,
parameter
COUNT_ENA
 =
1,
parameter
DATA_ZERO
 =
0,
parameter
ACK_ENA
 =
0,
parameter
RAM_TYPE
 =
"block"
) ( input rd_clk, input rd_rstn, input rd_en, output rd_valid, output [(BY
```

Wrapper to tie together fifo_ctrl, fifo_mem, and fifo_pipe.

Parameters

FIFO_DEPTH parameter	Depth of the fifo, must be a power of two number(divisable aka 256 = 2^8). Any non-power of two will be rounded up to the next closest.
BYTE_WIDTH parameter	How many bytes wide the data in/out will be.
COUNT_WIDTH parameter	Data count output width in bits. Should be the same power of two as fifo depth(256 for fifo depth... this should be 8).
FWFT parameter	1 for first word fall through mode. 0 for normal.
RD_SYNC_DEPTH parameter	Add in pipelining to read path. Defaults to 0.
WR_SYNC_DEPTH parameter	Add in pipelining to write path. Defaults to 0.
DC_SYNC_DEPTH parameter	Add in pipelining to data count path. Defaults to 0.
COUNT_DELAY parameter	Delay count by one clock cycle of the data count clock. Set this to 0 to disable (only disable if read/write/data_count are on the same clock domain!).
COUNT_ENA parameter	Enable the count output.
DATA_ZERO parameter	Zero out data output when enabled.

ACK_ENA parameter	Enable an ack when data is requested.
RAM_TYPE parameter	Set the RAM type of the fifo.

Ports

rd_clk	Clock for read data
rd_rstn	Negative edge reset for read.
rd_en	Active high enable of read interface.
rd_valid	Active high output that the data is valid.
rd_data	Output data
rd_empty	Active high output when read is empty.
wr_clk	Clock for write data
wr_rstn	Negative edge reset for write
wr_en	Active high enable of write interface.
wr_ack	Active high when enabled, that data write has been done.
wr_data	Input data
wr_full	Active high output that the FIFO is full.
data_count_clk	Clock for data count
data_count_rstn	Negative edge reset for data count.
data_count	Output that indicates the amount of data in the FIFO.

INSTANTIATED MODULES

pipe

```
fifo_pipe #(
                                                                    .
RD_SYNC_DEPTH(RD_SYNC_DEPTH),
                                                                    .
WR_SYNC_DEPTH(WR_SYNC_DEPTH),
                                                                    .
DC_SYNC_DEPTH(DC_SYNC_DEPTH),
                                                                    .
BYTE_WIDTH(BYTE_WIDTH),
                                                                    .
DATA_ZERO(DATA_ZERO),
                                                                    .
COUNT_WIDTH(COUNT_WIDTH)
) pipe ( .rd_clk(rd_clk), .rd_rstn(rd_rstn), .rd_en(rd_en), .rd_valid(s_rd_v
```

Pipe for data sync/clock issues.

control

```
fifo_ctrl #(
                                                                    .
FIFO_DEPTH(c_FIFO_DEPTH),
                                                                    .
BYTE_WIDTH(BYTE_WIDTH),
```

```
                                                                                .
 ADDR_WIDTH(c_PWR_FIFO),                                                        .
 COUNT_WIDTH(COUNT_WIDTH),                                                      .
 COUNT_DELAY(COUNT_DELAY),                                                      .
 COUNT_ENA(COUNT_ENA),                                                          .
 ACK_ENA(ACK_ENA),                                                              .
 FWFT(FWFT)
) control ( .rd_clk(rd_clk), .rd_rstn(rd_rstn), .rd_en(s_rd_en), .rd_addr(s_
```

Block RAM control, so it will act like a FIFO.

inst_dc_block_ram

```
 dc_block_ram #(
                                                                                .
 RAM_DEPTH(c_FIFO_DEPTH),                                                       .
 BYTE_WIDTH(BYTE_WIDTH),                                                        .
 ADDR_WIDTH(c_PWR_FIFO),                                                        .
 RAM_TYPE(RAM_TYPE)
) inst_dc_block_ram ( .rd_clk(rd_clk), .rd_rstn(rd_rstn), .rd_en(s_rd_mem_en
```

Block RAM

fifo_ctrl.v

AUTHORS

JAY CONVERTINO

DATES

2021/06/29

INFORMATION

Brief

Control block for fifo operations, emulates xilinx fifo.

License MIT

fifo_ctrl

```
module fifo_ctrl #(
parameter
FIFO_DEPTH
 =
256,
parameter
BYTE_WIDTH
 =
1,
parameter
ADDR_WIDTH
 =
1,
parameter
```

```
COUNT_WIDTH
 =
1,
parameter
GREY_CODE
 =
1,
parameter
COUNT_DELAY
 =
1,
parameter
COUNT_ENA
 =
1,
parameter
ACK_ENA
 =
0,
parameter
FWFT
 =
0
) ( input rd_clk, input rd_rstn, input rd_en, output [ADDR_WIDTH-1:0] rd_ad
```

Control block for fifo operations, emulates xilinx fifo.

Parameters

FIFO_DEPTH parameter	Depth of the fifo, must be a power of two number(divisable aka 256 = 2^8). Any non-power of two will be rounded up to the next closest.
BYTE_WIDTH parameter	How many bytes wide the data in/out will be.
ADDR_WIDTH parameter	Width of the RAM address bus to write data to.
COUNT_WIDTH parameter	Data count output width in bits. Should be the same power of two as fifo depth(256 for fifo depth... this should be 8).
GREY_CODE parameter	RAM address uses grey code instead of linear addressing.
COUNT_DELAY parameter	Delay count by one clock cycle of the data count clock. Set this to 0 to disable (only disable if read/write/data_count are on the same clock domain!).
COUNT_ENA parameter	Enable the count output.
ACK_ENA parameter	Enable ack on write.
FWFT parameter	1 for first word fall through mode. 0 for normal.

Ports

rd_clk	Clock for read data
rd_rstn	Negative edge reset for read.
rd_en	Active high enable of read interface.
rd_addr	Address to read data from in RAM.
rd_valid	Active high output that the data is valid.
rd_mem_en	Active high enable to read from RAM.
rd_empty	Active high output when read is empty.

wr_clk	Clock for write data
wr_rstn	Negative edge reset for write
wr_en	Active high enable of write interface.
wr_addr	Address to write data to in RAM.
wr_ack	Active high when enabled, that data write has been done.
wr_mem_en	Active high enable to write to RAM.
wr_full	Active high output that the FIFO is full.
data_count_clk	Clock for data count
data_count_rstn	Negative edge reset for data count.
data_count	Output that indicates the amount of data in the FIFO.

fifo_pipe.v

AUTHORS

JAY CONVERTINO

DATES

2021/06/29

INFORMATION

Brief

Pipe fifo signals to help with timing issues, if they arise.

License MIT

fifo_pipe

```
module fifo_pipe #(
parameter
RD_SYNC_DEPTH
 =
0,
parameter
WR_SYNC_DEPTH
 =
0,
parameter
DC_SYNC_DEPTH
 =
0,
parameter
```

```
BYTE_WIDTH
 =
1,
parameter
DATA_ZERO
 =
0,
parameter
COUNT_WIDTH
 =
1
) ( input rd_clk, input rd_rstn, input rd_en, input rd_valid, input [(BYTE_W
```

Pipe fifo signals to help with timing issues, if they arise.

Parameters

BYTE_WIDTH parameter	How many bytes wide the data in/out will be.
COUNT_WIDTH parameter	Data count output width in bits. Should be the same power of two as fifo depth(256 for fifo depth... this should be 8).
RD_SYNC_DEPTH parameter	Add in pipelining to read path. Defaults to 0.
WR_SYNC_DEPTH parameter	Add in pipelining to write path. Defaults to 0.
DC_SYNC_DEPTH parameter	Add in pipelining to data count path. Defaults to 0.
DATA_ZERO parameter	Zero out data output when enabled.

Ports

rd_clk	Clock for read data
rd_rstn	Negative edge reset for read.
rd_en	Active high enable input of read interface.
rd_valid	Active high output input that the data is valid.
rd_data	Output data input
rd_empty	Registered Active high output when read is empty.
r_rd_en	Registered Active high enable of read interface.
r_rd_valid	Registered Active high output that the data is valid.
r_rd_data	Registered Output data
r_rd_empty	Active high output when read is empty.
wr_clk	Clock for write data
wr_rstn	Negative edge reset for write
wr_en	Active high enable of write interface, feed into register.
wr_ack	Active high when enabled, that data write has been done, feed into register.
wr_data	Input data, feed into register.
wr_full	Active high output that the FIFO is full, feed into register.
r_wr_en	Register Active high enable of write interface.
r_wr_ack	Register Active high when enabled, that data write has been done.
r_wr_data	Register Input data
r_wr_full	Register Active high output that the FIFO is full.
data_count_clk	Clock for data count

data_count_rstn	Negative edge reset for data count.
data_count	Output that indicates the amount of data in the FIFO.

tb_cocotb.v

AUTHORS

JAY CONVERTINO

DATES

2024/12/10

INFORMATION

Brief

Test bench wrapper for cocotb

License MIT

tb_cocotb

```
module tb_cocotb #(
parameter
FIFO_DEPTH
 =
256,
parameter
BYTE_WIDTH
 =
1,
parameter
COUNT_WIDTH
 =
8,
parameter
```

```
FWFT
 =
0,
parameter
RD_SYNC_DEPTH
 =
0,
parameter
WR_SYNC_DEPTH
 =
0,
parameter
DC_SYNC_DEPTH
 =
0,
parameter
COUNT_DELAY
 =
1,
parameter
COUNT_ENA
 =
1,
parameter
DATA_ZERO
 =
0,
parameter
ACK_ENA
 =
1,
parameter
RAM_TYPE
 =
"block"
) ( input rd_clk, input rd_rstn, input rd_en, output rd_valid, output [(BY
```

Wrapper to interface with dut, FIFO

Parameters

FIFO_DEPTH parameter	Depth of the fifo, must be a power of two number(divisable aka 256 = 2^8). Any non-power of two will be rounded up to the next closest.
BYTE_WIDTH parameter	How many bytes wide the data in/out will be.
COUNT_WIDTH parameter	Data count output width in bits. Should be the same power of two as fifo depth(256 for fifo depth... this should be 8).
FWFT parameter	1 for first word fall through mode. 0 for normal.
RD_SYNC_DEPTH parameter	Add in pipelining to read path. Defaults to 0.
WR_SYNC_DEPTH parameter	Add in pipelining to write path. Defaults to 0.
DC_SYNC_DEPTH parameter	Add in pipelining to data count path. Defaults to 0.
COUNT_DELAY parameter	Delay count by one clock cycle of the data count clock. Set this to 0 to disable (only disable if read/write/data_count are on the same clock domain!).
COUNT_ENA parameter	Enable the count output.
DATA_ZERO parameter	Zero out data output when enabled.

ACK_ENA parameter	Enable an ack when data is requested.
RAM_TYPE parameter	Set the RAM type of the fifo.

Ports

rd_clk	Clock for read data
rd_rstn	Negative edge reset for read.
rd_en	Active high enable of read interface.
rd_valid	Active high output that the data is valid.
rd_data	Output data
rd_empty	Active high output when read is empty.
wr_clk	Clock for write data
wr_rstn	Negative edge reset for write
wr_en	Active high enable of write interface.
wr_ack	Active high when enabled, that data write has been done.
wr_data	Input data
wr_full	Active high output that the FIFO is full.
data_count_clk	Clock for data count
data_count_rstn	Negative edge reset for data count.
data_count	Output that indicates the amount of data in the FIFO.

INSTANTIATED MODULES

dut

```
fifo #(
                                                                    .
FIFO_DEPTH(FIFO_DEPTH),
                                                                    .
BYTE_WIDTH(BYTE_WIDTH),
                                                                    .
COUNT_WIDTH(COUNT_WIDTH),
                                                                    .
FWFT(FWFT),
                                                                    .
RD_SYNC_DEPTH(RD_SYNC_DEPTH),
                                                                    .
WR_SYNC_DEPTH(WR_SYNC_DEPTH),
                                                                    .
DC_SYNC_DEPTH(DC_SYNC_DEPTH),
                                                                    .
COUNT_DELAY(COUNT_DELAY),
                                                                    .
COUNT_ENA(COUNT_ENA),
                                                                    .
DATA_ZERO(DATA_ZERO),
                                                                    .
ACK_ENA(ACK_ENA),
                                                                    .
RAM_TYPE(RAM_TYPE)
) dut ( .wr_clk(wr_clk), .wr_rstn(wr_rstn), .wr_en(wr_en), .wr_ack(wr_ack),
```

Device under test,fifo

tb_cocotb.py

AUTHORS

JAY CONVERTINO

DATES

2024/12/09

INFORMATION

Brief

Cocotb test bench

License MIT

FUNCTIONS

random_bool

```
def random_bool()
```

Return a infinte cycle of random bools

Returns: List

start_clock

```
def start_clock(
dut
)
```

Start the simulation clock generator.

Parameters

dut Device under test passed from cocotb test function

reset_dut

```
async def reset_dut(
dut
)
```

Cocotb coroutine for resets, used with await to make sure system is reset.

single_word

```
@cocotb.test()
async def single_word(
dut
)
```

Coroutine that is identified as a test routine. This routine tests for writing a single word, and then reading a single word.

Parameters

dut Device under test passed from cocotb.

full_empty

```
@cocotb.test()
async def full_empty(
dut
)
```

Coroutine that is identified as a test routine. This routine tests for writing till the fifo is full, Then reading from the full FIFO.

Parameters

dut Device under test passed from cocotb.

in_reset

```
@cocotb.test()
async def in_reset(
dut
)
```

Coroutine that is identified as a test routine. This routine tests if device stays in unready state when in

reset.

Parameters

dut Device under test passed from cocotb.

no_clock

```
@cocotb.test()
async def no_clock(
dut
)
```

Coroutine that is identified as a test routine. This routine tests if no ready when clock is lost and device is left in reset.

Parameters

dut Device under test passed from cocotb.

tb_fifo.v

AUTHORS

JAY CONVERTINO

DATES

2021/06/29

INFORMATION

Brief

Test bench for fifo using fifo stim and clock stim.

License MIT

tb_fifo

```
module tb_fifo #(
parameter
IN_FILE_NAME
 =
in.bin,
parameter
OUT_FILE_NAME
 =
out.bin,
parameter
FIFO_DEPTH
 =
64,
parameter
```

```
 RAND_FULL
  =
 0
)()
```

Test bench for fifo. This will run a file through the system and write its output. These can then be compared to check for errors. If the files are identical, no errors. A FST file will be written.

Parameters

IN_FILE_NAME parameter	File name for input.
OUT_FILE_NAME parameter	File name for output.
FIFO_DEPTH parameter	Number of transactions to buffer.
RAND_READY	0 = no random ready. 1 = randomize ready.

INSTANTIATED MODULES

clk_stim

```
 clk_stimulus #(
                                                                                .
 CLOCKS(2),
                                                                                .
 CLOCK_BASE(1000000),
                                                                                .
 CLOCK_INC(1000),
                                                                                .
 RESETS(2),
                                                                                .
 RESET_BASE(2000),
                                                                                .
 RESET_INC(100)
) clk_stim ( .clkv({tb_dut_clk, tb_stim_clk}), .rstnv({tb_dut_rstn, tb_stim_
```

Generate a 50/50 duty cycle set of clocks and reset.

write_fifo_stimulus

```
 write_fifo_stimulus #(
                                                                                .
 BYTE_WIDTH(BYTE_WIDTH),
                                                                                .
 FILE(IN_FILE_NAME)
) write_fifo_stim ( .rd_clk(tb_stim_clk), .rd_rstn(tb_stim_rstn), .rd_en(~tb
```

Device under test WRITE stimulus module.

dut

```
 fifo #(
                                                                                .
```

```
 FIFO_DEPTH(FIFO_DEPTH),                                                    .
 BYTE_WIDTH(BYTE_WIDTH),                                                    .
 COUNT_WIDTH(8),                                                            .
 FWFT(0),                                                                   .
 RD_SYNC_DEPTH(0),                                                          .
 WR_SYNC_DEPTH(0),                                                          .
 DC_SYNC_DEPTH(0),                                                          .
 COUNT_DELAY(1),                                                            .
 COUNT_ENA(1),                                                              .
 DATA_ZERO(0),                                                              .
 ACK_ENA(0),                                                                .
 RAM_TYPE("block")
) dut ( .wr_clk(tb_stim_clk), .wr_rstn(tb_stim_rstn), .wr_en(tb_stim_valid),
```

Device under test, fifo

read_fifo_stimulus

```
 read_fifo_stimulus #(                                                      .
 BYTE_WIDTH(BYTE_WIDTH),                                                    .
 RAND_FULL(RAND_FULL),                                                      .
 FILE(OUT_FILE_NAME)
) read_fifo_stim ( .wr_clk(tb_dut_clk), .wr_rstn(tb_dut_rstn), .wr_en(tb_dut
```

Device under test READ stimulus module.